International Journal of Computer Science Engineering
and Information Technology Research (IJCSEITR)
ISSN(P): 2249-6831; ISSN(E): 2249-7943
Vol. 7, Issue 1, Feb 2017, 21-34
© TJPRC Pvt. Ltd.

INHERITANCE IN OBJECT ORIENTED PROGRAMMING EASIEST WAY TO TEACH AND LEARN INHERITANCE IN C++

TWINKLE PATEL

Department of Computer Science and Engineering, ITM Universe, Vadodara, Gujarat, India

ABSTRACT

When we are talking about Object-Oriented programming, the first strike of everyone's mind is object and class. Object and Class are core concepts to build Object-Oriented programming. C++ have some basic features: simple, easy to use, portable, syntax based, object based, robust, platform dependent and much built in libraries. There are some other basic features of Object Oriented Programming: Data abstraction, Encapsulation, Dynamic Binding, Inheritance and Polymorphism. Here we will see how to teach inheritance in C++ by different examples.

KEYWORDS: Inheritance, Object, Base Class, Derived Class, Private, Protected, Public & Construdtor

Received: Dec 20, 2016; **Accepted:** Jan 20, 2017; **Published:** Jan 25, 2017; **Paper Id.:** IJCSEITRFEB20174

INTRODUCTION

Inheritance is a basic feature of Object Oriented Programming language. It contains different classes; in which object of one class acquiring some or all property of another class. It is nothing but a hierarchical classification. In it we are creating one class that is *base class*, as we all know C++ supports reusability concept, so reusing the properties of that class to another class, which is *derived class*, or *child class*. You can also called base class as old class. And derived class is also known as *sub-class* [1].

For example, a parrot is a type of a flying bird; also an eagle is a type of a flying bird; a kiwi and penguin are type of a flightless bird. And these two flying and flightless birds are type of Birds. So here Bird is a base class as shown in figure 1.

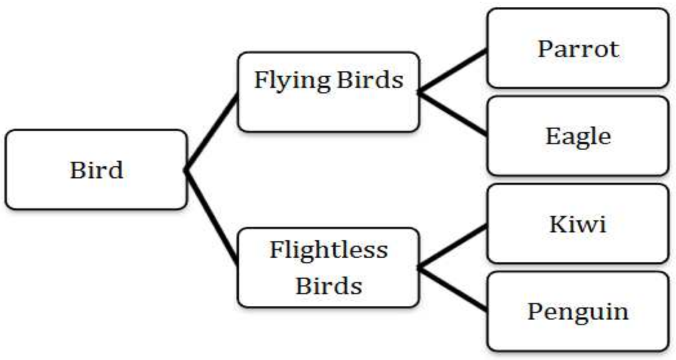

Figure 1: Basic Structure of Inheritance

Flying Birds and Flightless Birds are *child* class or *derived* class of *base* class Bird. The Parrot and Eagle both are of grandchild class of class Bird. But flying bird is a base class of class Parrot and Eagle. Kiwi and penguin are child class of base class flightless bird.

Syntax of Inheritance

```
class Derived Class Name : access-specifier Base Class Name
{
 //Body of Derived Class
};
```

We have different access-specifiers with different role: *public, private and protected.* If you want to match with our example, then you can see it works like below syntax. Assume only two classes are there, Bird and Flying Bird then syntax should be like this:

```
class Bird
{
};
class Flying : public Bird
{
};
```

To better understand of Inheritance we have different types of it. It will add additional features and helpful in many arenas.

TYPES OF INHERITANCE

Inheritance has various forms, which could be helping you to build much extensible and huge application [5]. You can apply any type of *inheritance* based on your requisites. Different forms of inheritance:

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

In this paper we will see all types of inheritance with syntax and example.

- **Single Inheritance**

The derived class with single base class is Single Inheritance from figure 2. In single inheritance, child class can acquire features of its parent class based on access-specifier. In this figure you can see two classes: shape and circle. Shape is base class of deriving class circle. Syntax of it:

```
class Base Class Name
{
};
class Derived Class Name: access-specifier Base Class Name
{
};
```

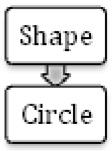

Figure 2: Single Inheritance

```
#include<iostream>
using namespace std;
class Shape
{
public:
Shape()
{
cout<<"Shape Called";
cout<<endl;
}
};

class Circle: public Shape
{
public:
Circle()
{
cout<<"Circle called";
cout<<endl; }
};
int main()
{
```

```
Circle cobj;

return 0;

}
```

You will get output after compiling and executing this program:

```
Shape Called

Circle Called
```

As we know when we create object of any class; constructor of that class automatically executes. As here the concept of inheritance means area of relationship between different classes at different level classes. Although the object of class Circle is created, we can able to execute constructor of parent class of it also. But you have to remember the order of it. Even though when you create object of derived class, it will first execute the constructor of its base class, then itself constructor print. Here shape is parent class, even though we create object of class circle it will execute shape first followed by circle.

- **Multiple Inheritance**

The derived class with several base classes is multiple inheritances. A derived class contains not only feature of single class but more than one class. For example, child has some features from his father and some features from his mother. So he acquires properties of both. As in figure 3

Syntax of It

```
class BaseClassName1

{

};

class BaseClassName2

{

};

class BaseClassName3

{

};

class Derived Class Name: access-specifier BaseClassName1, access-specifier Base-ClassName2, access-specifier BaseClass-Name3

{

};
```

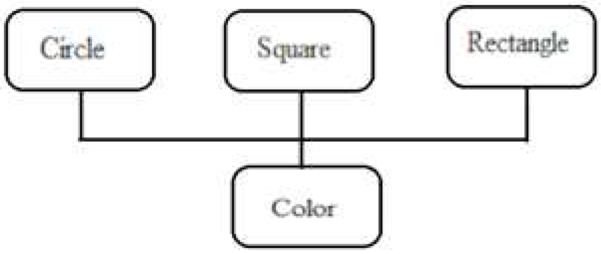

Figure 3: Multiple Inheritance

```
#include<iostream>

using namespace std;

class Circle

{

public:

Circle()

{

cout << "Circle's constructor called" << endl;

}

};

class Square

{

public:

Square()

{

 cout << "Square's constructor called" << endl; }

};

class Rectangle

{

public:

Rectangle()

{

cout << "Rectangle's constructor called" << endl;
}
};
class Color: public Circle, public Square, public Rectangle
{
public:
Color()
{
 cout << "Color's constructor called" << endl;
}
};
int main()
{
Color cobj;
return 0;
}
```

You will get output after compiling and executing this program:

```
Circle's constructor called
Square’s constructor called
Rectangle’s constructor called
Color’s constructor called
```

As we can see here we derivate all classes by public access

Specifiers. The Base classes Circle, Square and Rectangle have a constructor in their own classes and also in derived class. When we create object of derived class Color it prints output in a specific order as you can see. So class Color inherits the property from its parent class.

- **Multilevel Inheritance**

The deriving class inherits the property from another derived class this feature is known as *multilevel inheritance*.

The Syntax of It

```
class BaseClass1
{
};
```

```
class DerivedClass1: access-specifier BaseClass1
{
};
class DerivedClass2: access-specifier DerivedClass1
{
};
```

The Shape class is a base class; Circle is also a base class of class Color but child class of class Shape. Color is derived class of class Shape and class Circle as shown in figure 4

```
#include<iostream>
using namespace std;
class Shape
{
public:
```

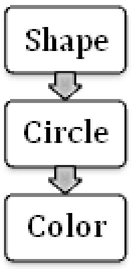

Figure 4: Multilevel Inheritance

```
Shape()
{
cout << "Shape's constructor called" << endl;
 }
};
class Circle: public Shape
{
public:
Circle()
{
cout << "Circle's constructor called" << endl;
}
};
class Color: public Circle
{
public:
Color()
{
cout << "Color's constructor called" << endl;
}
};
int main()
{
Color cobj;
return 0;
}
```

You will get output after compiling and executing this program:

```
Shape's constructor called
Circle's constructor called
Color’s constructor called
```

- **Hierarchical Inheritance**

The feature of one class inherited by more than one class. There are the possibilities in which you can find feature of base class in more than one derived class.

Syntax of it is

```
class BaseClass1
{
};
class DerivedClass1: access-specifier
BaseClass1
{
};
```

```
class DerivedClass2: access-specifier BaseClass1
{
};
class DerivedClass3: access-specifier BaseClass1
{
};
```

This Concept is More Helpful in Projects of IT Sectors

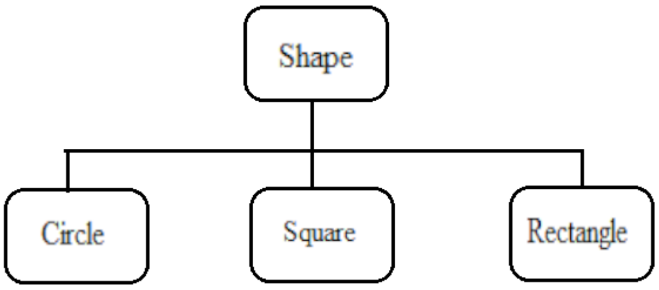

Figure 5: Hierarchical Inheritance

The Circle class, Square class and Rectangle class are derived class from base class Shape as in figure 5

```
#include<iostream>
using namespace std;
class Shape
{
public:
Shape()
{
cout << "Shape's constructor
 called" << endl;
}
};
class Circle: public Shape
{
public:
Circle()
{
```

```
cout << "Circle's constructor called" << endl;
}
};
class Square: public Shape
{
public:
Square()
{
cout << "Square's constructor called" << endl;
}
};
class Rectangle: public Shape
{
public:
Rectangle()
{
 cout << "Rectangle's constructor called" << endl;
}
};
int main()
{
Rectangle robj;//print first
Square sobj; //print second after
 rectangle
Circle cobj; //print third after
 square
return 0;
}
```

You will get output after compiling and executing this program:

```
Shape's constructor called

Rectangle's constructor called

Shape's constructor called

Square's constructor called

Shape's constructor called

Circle's constructor called
```

In this example we can see there is no dependency among derived classes.

- **Hybrid Inheritance**

This mechanism provides all flexibility means it is the mixture of two or more previous forms of inheritance

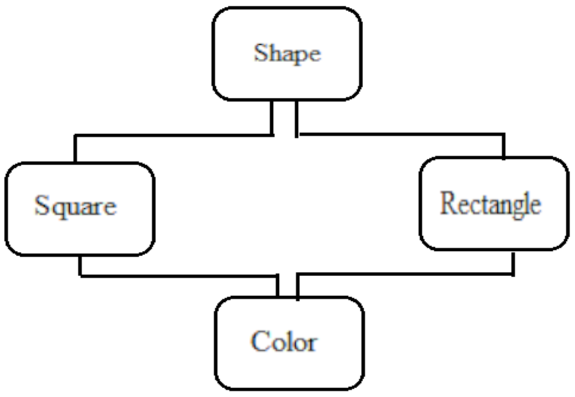

Figure 6: Hybrid Inheritance

```
#include<iostream>

using namespace std;

class Shape

{

public:

Shape()

{

 cout << "Shape's constructor called" << endl;

}

};

class Square: public Shape

{
```

```
public:
Square()
 {
cout << "Square's constructor called" << endl;
}
};
class Rectangle: public Shape
{
public:
Rectangle()
{
cout << "Rectangle's constructor called" << endl;
}
};
class Color: public Square, public Rectangle
{
public:
Color()
{
cout << "Color's constructor called";
cout << endl;
}
};
int main()
{
Color cobj;
return 0;
}
```

You will get output after compiling and executing this program:

```
Shape's constructor called

Square's constructor called

Shape's constructor called

Rectangle's constructor called

Color's constructor called
```

CONCLUSIONS

This paper presents an idea about inheritance and how to teach inheritance in classroom. It is also useful to learn for students by own self. Here I also covered idea about different types of forms of Inheritance with example. So that every one can understand easily.

REFERENCES

1. *E Balagurusamy, "Object Oriented Programming with C++", 6th Ed.,New Delhi, TATA McGraw-Hill.*
2. *Krishnaprasad Thirunarayan, "Inheritance in Programming Languages"*
3. *Budd, T. (2002) Introduction to Object-Oriented Programming, Third Edition, Addison-Wesley.*
4. *Markku Sakkinen, "Inheritance and other main pinciples of C++ and other object oriented language",1992*
5. *Robert Lafore, "Object Oriented Programming in Turbo C++", 4th Ed*